Name: XU Xiaoming (HT050666A)
Degree: B.Sc.(Hons.) & B.Eng.(Hons.)
Dept: School of Computer Engineering, Wenzhou University, PRC
Thesis Title: On the Effect of Congestion-Induced Surfer Behavior

Abstract

The main scope of this thesis is the effect of user behavior in web traffic modeling and related research. And we focus on a recently presented traffic model called "Surfer Model" and the related research. The main contribution here is to quantify the original surfer model in some level so as to use it in other research experiments, hoping to find difference caused by explicitly taking user behavior into account. The work includes three major aspects: learning from traces, constructing surfer simulator, and using the simulator to study the effects of user behavior.

Keywords: Network traffic modeling, Surfing session
Congestion, Open model, Queue, RED

On the Effect of Congestion-Induced Surfer Behavior

by
XU Xiaoming

Department of Computer Science
School of Computing
National University of Singapore

2006/2007

On the Effect of Congestion-Induced Surfer Behavior

XU Xiaoming (HT050666A)
(B.Sc.(Hons.) & B.Eng.(Hons.), Wenzhou University, PRC)

A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE
SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE

2007

Acknowledgements

First of all, I would like to acknowledge my supervisor Professor Tay Yong Chiang's kind and invaluable suggestions in helping me complete this project. I also would like to take this opportunity to express my gratitude to all those who have helped me through this project in Communication and Internet Research Lab (CIRL), School of Computing.

Contents

List of Figures

Abstract

The main scope of this thesis is the effect of user behavior in web traffic modeling and related research. And we focus on a recently presented traffic model called "Surfer Model" and the related research. The main contribution here is to quantify the original surfer model in some level so as to use it in other research experiments, hoping to find difference caused by explicitly taking user behavior into account. The work includes three major aspects: learning from traces, constructing surfer simulator, and using the simulator to study the effects of user behavior.

Chapter 1

Background

This chapter gives a brief review of data network traffic modeling, focusing on Web traffic models, and leads to the motivation of our new traffic model.

1.1 An overview of data network traffic modeling

The development of data networking in the 1960s, first as an academic exercise, and subsequently and rapidly as a means of providing a host of new services, presents new challenges and opportunities to the well-established field of tele-traffic theory. Apart from a few pioneering investigations, such as Mandelbrot [1], some of the distinguishing features of data traffic from voice telephony were noticed as early as the late 1980s (Fowler and Leland [2], Meier-Hellstern et al. [3]). In their pioneering study of LAN traffic, Willinger et al. [4] presented data and argued for use of alternative models. By showing that sufficiently aggregated data traffic exhibited self-similarity over a wide range of time scales, the authors argued for the use of fractal models and more explicitly the use of statistical processes exhibiting long-range dependence (LRD). In subsequent reports [5], the authors contended that the underlying cause of self-similarity was effectively unrelated to the mechanisms of data transmission, and was exclusively due to the nature of aggregated load. Unlike voice, individual streams of load in data networks followed distributions with heavy tails, the aggregation of

which, it was argued, gave rise to its observed self-similarity. Cox gave a different ($M/G/\infty$) theoretical model of long-range dependent process. (details in [6])

1.2 Simple statistical models

For analytic simplicity, it is a good thing if we can model the traffic with a single statistical distribution. Many early efforts are made in this aspect.

Poisson process might be the most popular model for the WAN traffic in the early days (before 1995), attributed to its success in telephone traffic and its attracting properties such as memoryless (for service time with exponential distribution, the additional time needed to complete a customer's service in progress is independent of the time when the service started), merging (if two or more independent Poisson process are merged into a single process, the merged process is a Poisson process with a rate equal to the sum of the rates), splitting (if a Poisson process is split probabilistically into two processes, the two processes to be obtained are both Poisson), etc.

However, in their milestone work [7], Paxson and Floyd prove that in many cases Poisson process seriously underestimates the burstiness of TCP traffic over a wide range of time scales thus is not capable to be used to model WAN. Years later, Anja Feldmann proves that TCP connection arrivals can be modeled as self-similar process (details in [8]). All of the above results are based on wide area network trace.

The above-mentioned traffic models try to capture the general characteristics of WAN traffic. Though simple (in a sense) and powerful (mathematically), they are not accurate enough to regenerate the traffic. To provide practical traffic models that can be directly used by network traffic engineers, recent traffic models become more and more specific.

1.3 Web traffic models

From late 1990s, web/Http traffic began to dominate the traffic on Internet. As a result, a lot of work has been done to trace and model the web traffic for both server and link performance using different techniques.

In this section, we do a survey on existed web traffic models and introduce several widely referenced works.

1.3.1 Abrahamsson & Ahlgren's

Abrahamsson and Ahlgren model a web client using empirical probability distributions for user clicks and transferred data sizes. By using a heuristic threshold value to distinguish user clicks in a packet trace they get a simple method for analyzing large packet traces in order to obtain information about user OFF times and amount of data transferred due to a user click [9].

They use the threshold value 1 second to separate connections that belong to different web pages. The main reason for the choice of this value is that users will generally take longer than one second to react to the display of a new page before they order a new document retrieval. In order to validate the method and threshold value, a proxy X-server is used to log time-stamps on the mouse button-up events when using Netscape. The logged clicks are then compared with the click sequence computed from tcpdump trace with the method described above.

Comments

The problem of A&A's model is that they just extract and sum up all the properties from the trace, not use any statistical distributions to model the traffic. The result of this pure empirical method is thus no more than reproducing their traced traffic.

HTTP request sizes show a bimodal distribution.
HTTP reply sizes have a heavy-tailed distribution, and tend to be larger than request sizes.
A simple heuristic based on timing can be used to group individual files into documents.
The number of files per document tends to be small; 80% of documents required less than four file transfers.
HTTP requests to retrieve the first file of a multi-file Web page tend to be longer than subsequent requests.
The first file of a Web page tends to be larger than subsequent files.
The number of consecutive documents retrieved from a given server tends to be small. 80% of visits to a server's document space resulted in fewer than six documents being retrieved.

Figure 1.1: Selected measurement results

1.3.2 Mah's

Back in 1997, Mah published a paper presenting a more mathematical and practical web traffic model which is widely used later.

They used the tcpdump packet capture utility, running on a DEC Alpha 3000/300, to record TCP/IP packet headers on a shared 10Mbps Ethernet in the Computer Science Division at the UC Berkeley, during four periods in late 1995.

The subnet examined is a stub network (no transit traffic) with approximately one hundred hosts; almost all are UNIX workstations used principally by a single user, with several web servers associated with different research groups.

The model is defined by several key parameters of web traffic. And the distributions of parameters extracted from the traces are matched with variant statistical distributions. A summary of major results is given in Fig.1.1[10].

Comments

Mah's model extracts major statistical properties of web traffic. But it does not abstract the curves into classic distributions. For example, he concludes that the reply sizes have a heavy-tail distribution, yet does not analyze which classic heavy-tail distribution models it best and what values should be assigned to the parameters. Besides, the model implicitly assumes that user's behavior remains the same in different network conditions.

1.3.3 Choi&Limb's

Choi and Limb present a more sophisticated and accurate Web traffic model [11]. In particular, more application related traffic parameters are taken into account. The methodology is following.

Traffic trace: Use tcpdump to catch and record TCP header as well as 300 bytes payload which should be enough to cover HTTP header so as to support bi-layer trace.

Definition and detection of a web request: They use web-request as analysis unit rather than a web page so as to accurately catch properties of web traffic, since new web techniques such as auto-update and multi-connection download have made a web page inappropriate for base unit of analysis. (The purpose of which is basically the same with click detection in [9]) A web-request is a page or a set of pages resulting from a request by a user. By Choi's definition, (1) a web-request is initiated by a human action and (2) the first object in a web-request is an HTML document. Consequently, a request becomes a web-request if it is used to request for an object whose extension contains either ".html", ".asp", ".cgi" or "/" (/ implies "index.html" in the directory, also becomes a web-request); or if it results in a response of MIME type "text/html" and the HTTP status code 200 (OK).

Data parsing: Trace data is passed to parser script written with Perl. The start and end times of the web-requests are then recorded and used to parse parameters in the TCP trace. Empirical distributions of HTTP-layer parameters are obtained. In the TCP trace, the parser script parses TCP-layer parameters based on the boundaries that are recorded

Parameters		Mean	Median	S.D.	Best-fit
Request size		360.4	344	106.5	LN
Object size	Main	10710	6094	25032	LN
	In-line	7758	1931	126168	LN
Parsing time		0.13	0.06	0.187	G
Number of In-line objects		5.55	2	11.4	G
In-line Inter-Arrival time		0.86	0.17	2.15	G
Viewing (OFF) time		39.5	11.7	92.6	W
Number of Web-requests	Non-cached	12.6	5	21.6	LN
	Cached	1.7	1	1.7	GM

Figure 1.2: Summary statistics for HTTP parameters (LN=Lognormal, G=Gamma, W=Weibull and GM=Geometric)

when HTTP-layer parameters are parsed.

Model building: Once empirical distributions of the individual parameters are obtained, compare each distribution with different standard probability distributions and select the best fit. The Quantile-Quantile plot (Q-Q plot) is used to test the fit of the data to the model. If the model fits the data perfectly then the plotted points lie on a straight line. The best standard probability distribution is determined to be the one that minimize the root-mean-square of the deviation from a straight line. The candidates used here are Weibull, Lognormal, Gamma, Chi-square, Pareto and Exponential (Geometric) distributions.

The resulting statistics of HTTP parameters are summarized in Fig. 1.2[11].

Choi&Limb's model validation

To validate the model, they implemented a traffic generator which simulates an ON/OFF source (see section III in [11]). The state transition diagram of the traffic generation process is show in Fig. 1.3[11]. Two measurements from the synthesized traffic trace that are independent of any measurements used in constructing the model are used to validate the model. They are on-time and variation of the required bandwidth in time.

On-time from both the model and the trace matches a Weibull distribution with the shape

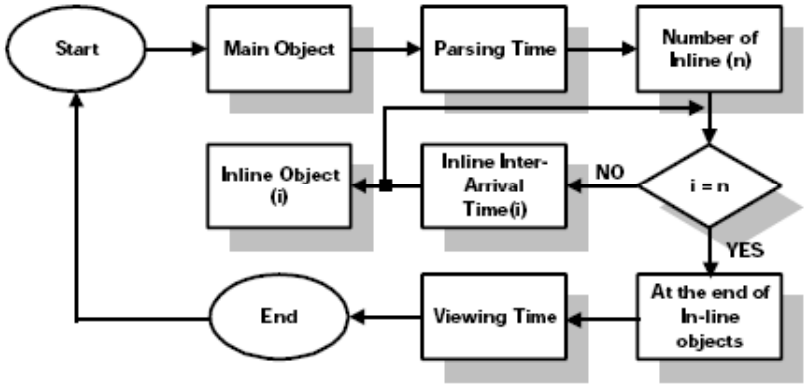


Figure 1.3: State transition diagram for Web traffic generation.

parameters 0.77 and 0.68, respectively. The mean and standard deviation of the traces are 11.34 and 23.85 and those of the model are 10.49 and 20.33. The cumulative density function comparison is shown in Fig. 1.4[11].

For variation of required bandwidth in time, they recorded the sum of bytes in ten millisecond granularity from the trace and model. In order to compare the overall behavior, ten seconds granularity is used in plotting. The means of the required bandwidth of the model and those of the trace closely match as shown in Fig. 1.5[11].

Comments

This model is more elaborate than Mah's. It parses web traffic into more parameters, and gives best-fit distributions for them.

Yet the model shares a common shortage with all the web traffic models discussed above. They implicitly assumes that user's behavior remains the same in different network conditions, which is intuitively not true.

1.4 Motivation to the surfer model

When we surf the web, we always expect the desired pages to be loaded in a reasonable period (usually several seconds). If the loading of the requested data is delayed by the environment

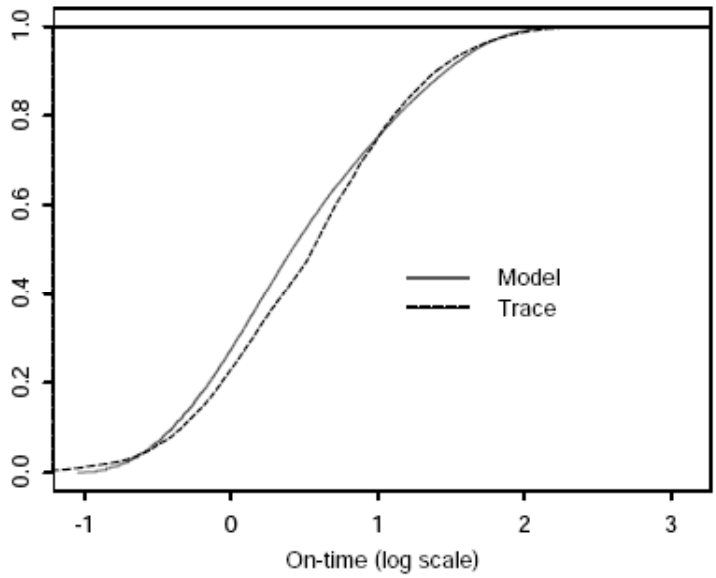


Figure 1.4: CDF comparison of On-times of Trace and On-time of Model. X-axis is log-scaled.

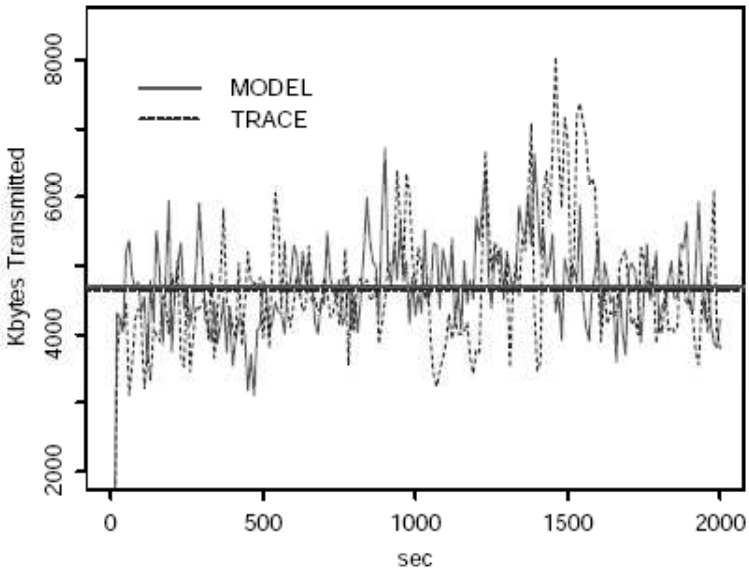


Figure 1.5: The variation of the demanded band-width in time. Two parallel lines indicate the mean of samples.

factors, e.g. network congestion, we are likely to refresh the page or switch to other ones. When we face a long waiting time for many different web pages (from different sites), we might guess there is something wrong with the network and are likely to stop the surfing session. This way, network traffic is affected and changed by the congestion condition.

More formally, a web surfer reacts to congestion in two ways:

(U1) She may abort a slow download by clicking "Stop", "Reload", "Refresh" or another hyper-link.

(U2) She may cut short her surfing session.

Tay et al. named such behavior congestion-induced **user backoff**. User backoff is important both for network stability and network management. It can throttle traffic from a flash crowd and mooth out self-similar traffic, thus possibly makes elaborate traffic engineering for countering burstiness unnecessary at all!

1.5 Contribution and content introduction

My main contribution in this project is as follows. I quantified the original surfer model in some level through learning from the traces, making it possible to emulate congestion-induced user behavior (section 3.1 and 3.2); and then constructed a simulator based on the model (section 3.3); after that, I used the simulator to study the effect of user behavior on the performance of RED, and proved our hypothesis that ignoring congestion-induced user behavior might lead to incorrect conclusions (section 4.2 to 4.4).

Chapter 1 gives a short survey of related work and the motivation of a new traffic model. Chapter 2 briefly introduces the innovative surfer model and my contribution. Then, chapter 3 begins with choosing good congestion indicator and related rationality, discusses my related research results, and explains choices that we made when constructing the simulator. In chapter 4, we use the constructed simulator to study the effect of surfer behavior on RED performance. Finally, chapter 5 gives possible future work to do.

Chapter 2

Surfer Model

This chapter introduces Tay's surfer model and leads to my work based on it.

2.1 User session model

We assume the user is a web surfer who reacts to congestion, and focus on HTTP flows and user backoff. Our model groups HTTP requests into sessions. Here a surfing session is defined as a period starts from the first click/typing in a web browser, and ends with the closure of the web browser or user leaving. (Tay's more complex model will count non-reactive users and UDP flows in.)

For now, we adopt an open model (the session arrival rate $r_{session}$ is constant, regardless of congestion), since it is a bit simpler but enough for the demonstration in this paper. (Tay has removed the assumption of constant $r_{session}$ in his closed model.) Besides, if we restrict the time span to a few minutes, the assumption of constant $r_{session}$ is reasonable.

In each session, a user sends HTTP requests with clicks on bookmarks, hyper-links, submit buttons, etc. For convenience, typing in an URL is considered as a click too. Let r_{click} be the click rate. Note that in general, each click generates multiple (in series, or parallel) HTTP request-responses. The traffic they send to the user is called a download (equivalently, Web object or Web request in some literature).

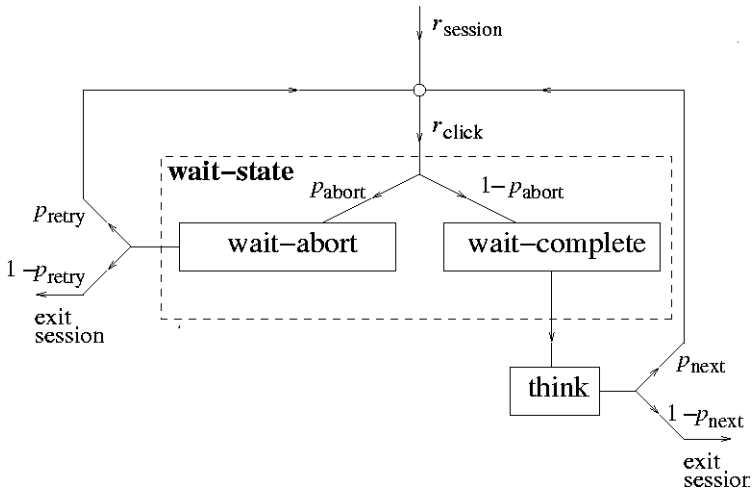


Figure 2.1: Surfer's session model: p_{retry} is the proportion of aborted downloads that are followed by another click in the session, and p_{next} is the proportion of completed downloads that are followed by another click in the session.

After a click, a surfer enters a **wait** state. If the wait is too long, she may abort the download. Let p_{abort} be the proportion of downloads that are aborted. This behavior can be modeled by splitting the wait state into a **wait-abort** state for aborted downloads and a **wait-complete** state for completed downloads. Let p_{retry} be the proportion of aborted downloads that are followed by another click in the session and p_{next} the proportion of completed downloads that are followed by another click in the session. Fig. 2.1 [12] shows the resulting session model.

For the detailed mathematical model, please refer to Tay et al.'s paper [12].

2.2 Practical meaning of this new model

Tay et al.'s new model differs from traditional web traffic models by explicitly taking congestion-induced user behavior into account. And some evidence has suggested that users really would react to congestion ([12]). However, quite a lot of researches have been done ignoring this kind of user behavior. As a result, it should be meaningful to review these research to see if their results have been misled by ignoring this user element.

2.3 Limitation and extension to be made

The surfer model introduced in this chapter and in [12] is still a qualitative model, whose meanings would be limited if it cannot be used in other researches (say, simulation research) to predict the real traffic more accurately. Thus in this thesis, we try to find some quantitative relations for the model and use these relations in our surfer simulator.

The rest of this paper focuses on how to construct a simulator that can well imitate congestion-induced user behavior of a group of surfers, and how we use it to study the effects of this behavior with the simulator. Work described in the following chapters is all done by the author of the thesis unless indicated otherwise.

Chapter 3

From Model to Simulator

Our goal in this chapter is to construct a surfer simulator Since quantifying the surfer model and constructing the surfer simulator are closely related. We put them together here in one chapter. First, we should refine the raw traced data before learning. Second, we must find mathematical relations between user behavior and other parameters, i.e. how do what things lead user behavior. Third, we will imitate user behavior in the real life with the learned relations.

Constructing a simulator which can imitate user's behavior is non-trivial. Many challenges are met during the procedure. 1) We need to filter out noises in the raw data and adjust inaccurate data that are caused by limitation of protocols; 2) Before learning mathematical functions, we need to choose a good congestion indicator from many candidates, and the choice may affect the difficulty of building our surfer simulator; 3) We have to imitate user behavior in the real life with only a few mathematical relations. We will explain in details when we meet those challenges one by one.

The trace used in the learning part of our work is a 50GB tcpdump trace obtained from a link in an academic network over two work days by Tran et al.

3.1 Analyzing packet traces

First, we need to group packets into downloads. A **download** is a collection of request-response pairs, the first of which is initiated by a click. Our method is to check the referrer field of HTML objects, and to use them as pointers to main pages recursively so as to group them into downloads. Meanwhile, other important information about the downloads such as the final status (completed or aborted) should be logged. The packet trace parsing is done by Tran with a tool named SAX. For details of SAX (e.g. how to judge the final status of a download), see [13].

Then we group downloads into sessions. Our method is to check idle times between downloads that are initiated by the same users. If the idle time exceeds a threshold, it is treated as a session break. The threshold that we use is 10 minutes (600 seconds). We filter out clients who have more than 2000 downloads, because they are unlikely to be real users.

To simulate the user behavior, we need not only downloaded file sizes, but also expected file sizes (we will explain the reasons in later sections); and not only the real download times, but also the expected download times. For many reasons, the expected file sizes logged in the traces are not accurate. For example, in some records, the downloaded sizes are equal to the expected sizes, yet the downloads still aborted (the completed flag is false). The reason could be that the user aborted when some headers of objects had not been received. In some cases, the downloaded file size can even be larger than the logged expected size. Apparently, using the logged expected size to calculate the expected download time may not be correct. To deal with those 'abnormal' records, we adjust the file size field of the records, whose downloaded size is larger than or equal to its expected size, in the following way: $S_{expect} = S_{expect} + S_{down}$ (where S_{expect} is the expected file size, S_{down} is the downloaded file size, '=' means assigning). This is not a perfect way and is supposed be rectified soon in the continuing research.

3.2 Learning parameters and their relations

The most important relations between user behavior and network environment in our model are the relations between the three probabilities ($P_{complete}/P_{abort}$, P_{next}, and P_{abort}, short as Pc/Pa, Pn, Pr) and the selected congestion measurement parameter. To measure these probabilities, let n_{click}, $n_{complete}$, n_{abort}, n_{next}, n_{retry} be (respectively) the number of downloads, completed downloads, aborted downloads, clicks after think time, and retries after aborts. Then, one can calculate the probabilities by

$$P_{complete} = \frac{n_{complete}}{n_{click}}, P_{abort} = \frac{n_{abort}}{n_{click}}, P_{next} = \frac{n_{next}}{n_{complete}}, P_{retry} = \frac{n_{retry}}{n_{abort}}. \tag{3.1}$$

In [12], which introduces the surfer model, the number of concurrent downloads k is used as the congestion indication parameter. Thus we studied $P(k)$s ($P(k)$ is to view the three probabilities as functions of variable k) first.

3.2.1 The P(k) relations

The first problem we meet is how to reasonably calculate the k. Currently there is no standard way to do that. Two reasonable methods are in mind. Both are tried while studying the P(k) relations.

The common part of the two ways is that they are both weight-based. We split the continuous time into thousands of 1-minute-long slots, and assign a k-weight value to every time slot. The k-weight of a time slot is the expected number of concurrent downloads that a download runs with when passing the slot. So if a download's life time overlaps a time slot, the k-weight of the time slot increases $overlapped\_length/length\_of\_timeslot$ (See Fig.3.1, where $W_n = (x_1 + x_2 + x_3 + x_4)/\Delta$). We can loop through the downloads to obtain the k-weight of all the time slots.

A naive way of P(k) calculation would be using k-weight as the k value of a time slot, and

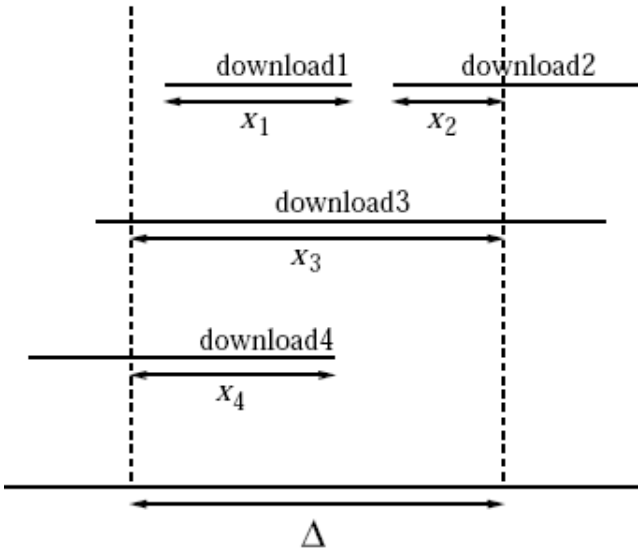


Figure 3.1: Calculate K by time weight

counting all the numbers of downloads, aborts etc. in the time slot as the numbers that are to be used to calculate respective probabilities. This way does not work for several reasons: (1) How do we count the downloads in a particular time slot? Counting the starts, the ends or both? (2) The downloads that overlap this time slot may be affected by other time slots that they overlap on. (3) Special cases like $P_{abort} = 1$ or $P_{abort} = 0$ can easily happen.

Another naive way is to use session as basic unit for counting k values and probabilities, which also has some awkward properties. For example, conditional probabilities like Pn and Pr will be ill-defined if $n_{abort} = 0$ or $n_{complete} = 0$ (number of aborts is 0, or number of completes is 0). Thus we have to use more tricky ways to retrieve the P(k) relations.

Our methods for calculating P(k)s

Our first way keeps using time slot as basic unit, as when we calculate the k-weights. For every time slot, we filter out the sessions that intersect with the slot, and then sum up all the numbers of completes, aborts etc. that happen in the time span from the start time of this slot to the end of those sessions, to calculate probabilities. Thus every time slot has a (k-weight, prob) pair, with which we can obtain the three P(k) curves. By counting from the

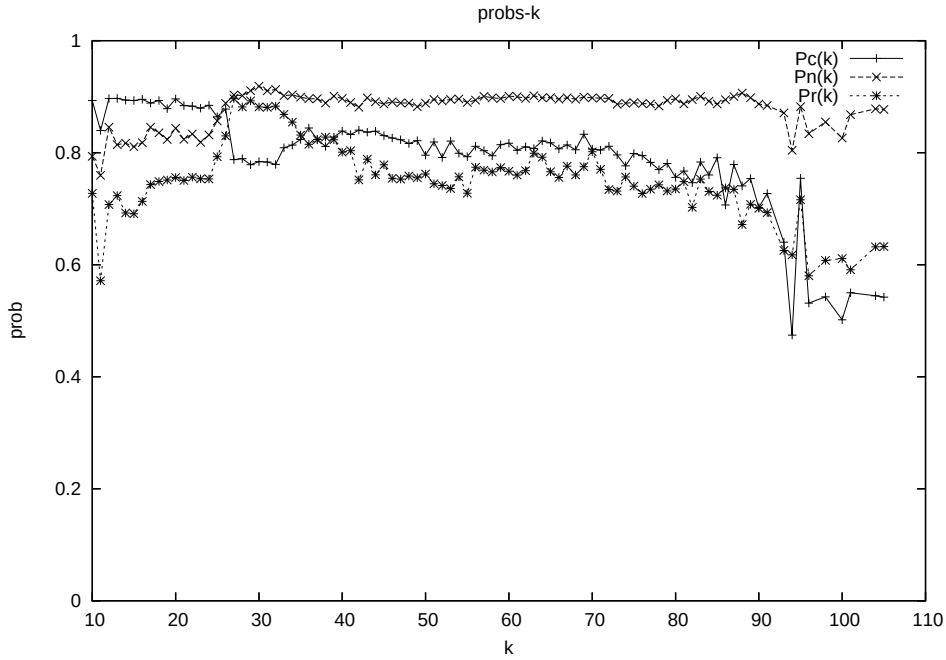


Figure 3.2: P(k) relations using time slot as basic unit.

start of a time slot to the end of sessions, we accumulate the effect of congestion in former time slot to sessions that extend to later time slots and overcome other problems that we mentioned above.

Fig.3.2 gives the corresponding relations that we have learned. It shows that Pc and Pr drops with time-slot based k, while Pn is generally stable with it.

Our second way uses download as basic unit. For every download, we check the k-weights of the time slots it intersects with, calculate the weighted average of k as the k value of that download. And the final status of the download is still its logged status. After looping through all the downloads, we can group the downloads by integer fraction of their k values, and then count the numbers of aborts, completes etc. in each k group to obtain respective probabilities. To illustrate the calculation of download based k, we give a numeric example with assistant of Fig.3.3. In the figure, S1 and S2 are sessions (not all sessions are drawn); D1 and D2 are downloads; T is the length of a time slot; W_i (i=1...5) are the k-weights of each time slot. With the download based method,

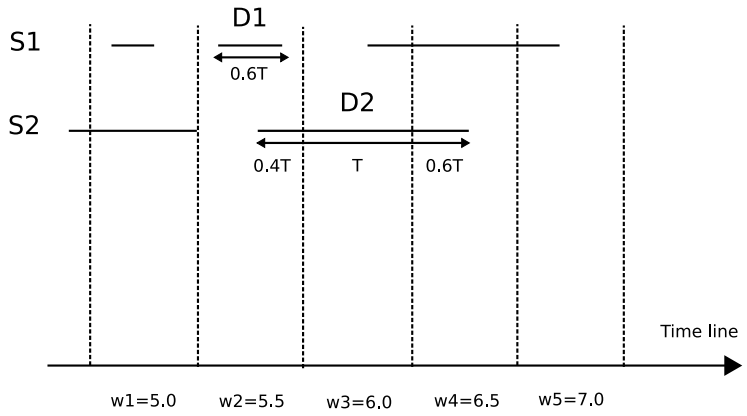


Figure 3.3: Illustration of k calculation

$$K_{D1} = W_2 = 5.5 \tag{3.2}$$

, since it only lives in slot 2; while

$$\begin{aligned} & K_{D2} \\ = \; & (0.4 * W_2 + 1.0 * W_3 + 0.6 * W_4)/(0.4 + 1.0 + 0.6) \\ = \; & (0.4 * 5.5 + 1.0 * 6 + 0.6 * 6.5)/(0.4 + 1.0 + 0.6) \\ = \; & 6.05 \end{aligned} \tag{3.3}$$

By focusing on every particular download, we get the effects of all the other concurrent downloads that intersect with its life time. And since the download is the basic unit, we use its original status to count the probabilities, thus we also have wiped away the special case problem mentioned above. This way seems to be fairer in calculating k (concurrent number of downloads) for each particular download. But it is hard to be used to predict the three

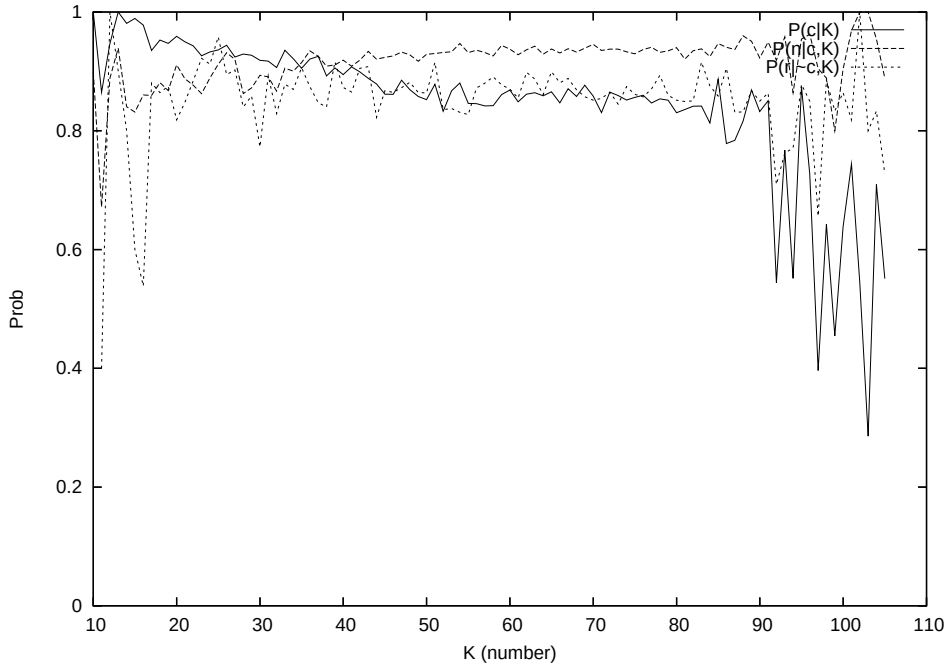


Figure 3.4: P(k) relations using download as basic unit.

probabilities of a download, since we need to sum up all the k-weights of the time slots the download passes.

Fig.3.4 gives the corresponding relations that we have learned. It shows a similar trend of the three P(k)s in the middle range, with big fluctuations in head and tail part. Fig.3.5 gives a reference of the sample numbers in our learning procedure of download based P(k)s, which shows that we have enough samples in the range of $20 \leq k \leq 90$. The lack of samples could be the reason of the fluctuations for small and large k. In contrast, the time-slot based method counts from start of a time-slot to the end of all intersected sessions, thus shows more stable curves.

3.2.2 The P(T) and P(Te) relations

Another choice is to use download time as congestion measurement parameter. This method is quite intuitive for web surfers. When one page is likely to take a very long time to load the user is likely to abort (refresh or stop). We have two candidates here: the time spent in

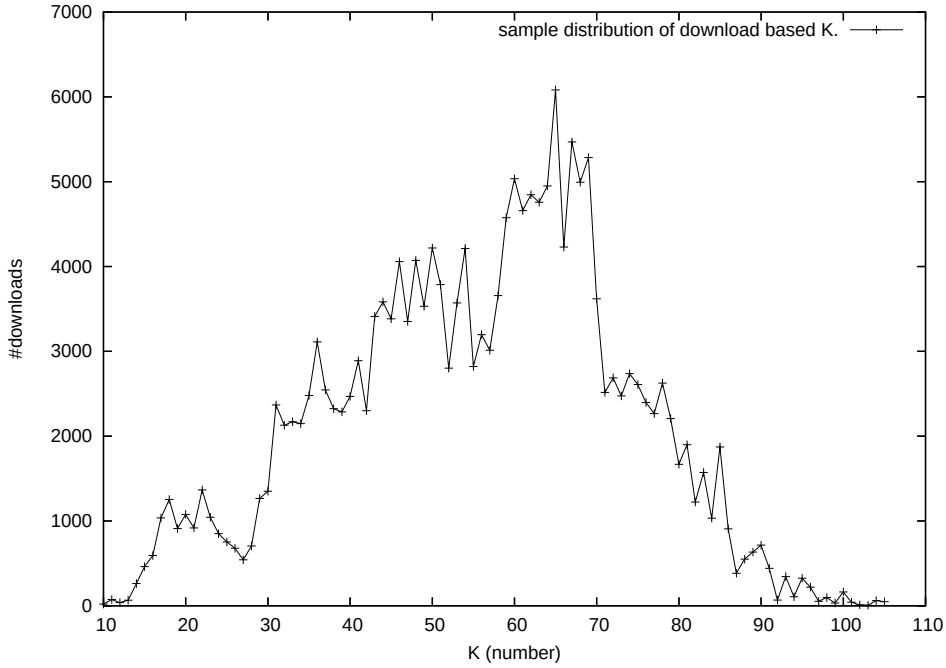
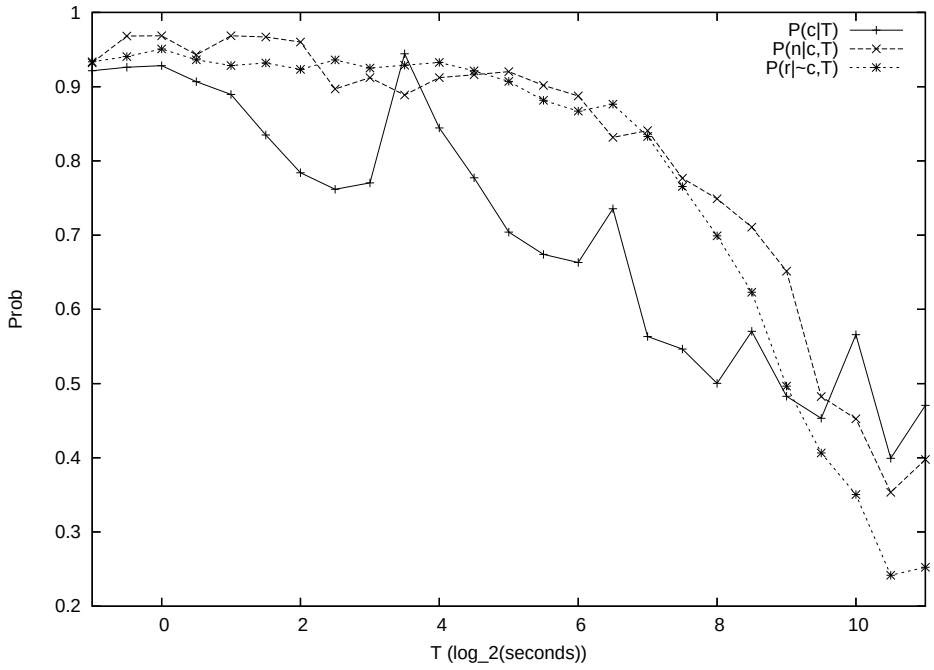


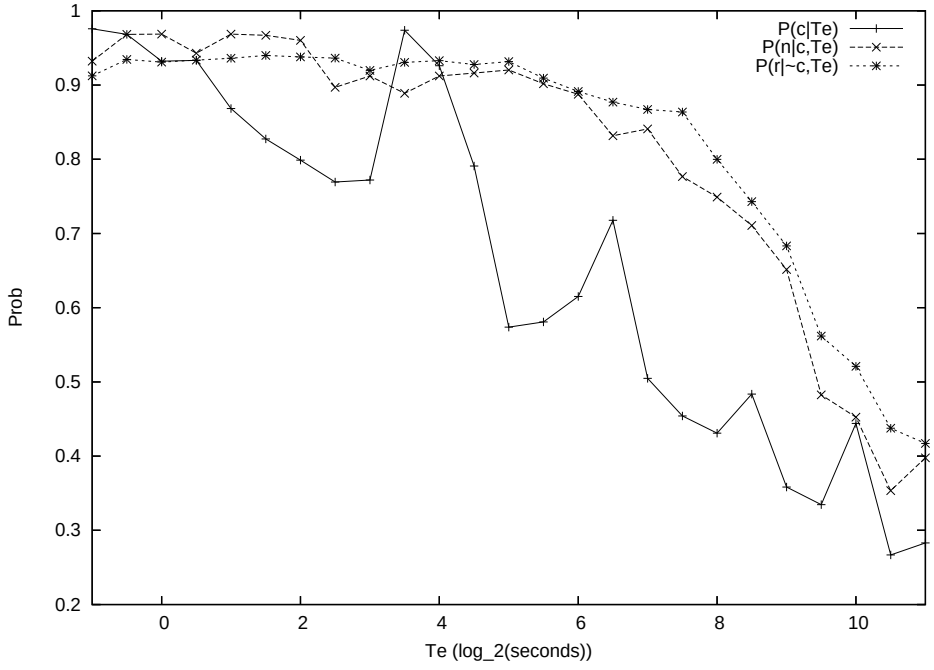
Figure 3.5: Samples distribution of P(k) relations using download as basic unit.

downloading (no matter eventually completed or not) T, and the expected download time Te. The ideal case is, the longer the expected download time is, the more likely the user would abort.

Some people would argue that those users who are downloading big files (like setup files or video clips) would be more patient than those who are just loading Web pages. But our research reveals that except those extremely large video downloads, the complete rate of the downloads decreases as the expected download time increases. The curves in Fig.3.6(a) and Fig.3.6(b) are learned from our trace summary, which contains many downloads whose sizes are larger than 100MB. We can see that the probability curves drops nicely when T and T_e is less than 2000 seconds. Fig.3.7 gives a reference of the sample numbers in our learning procedure of P(Te)s, which shows that we have hundreds of samples even for the very large and long term downloads.



(a)



(b)

Figure 3.6: Times vs. Probabilities

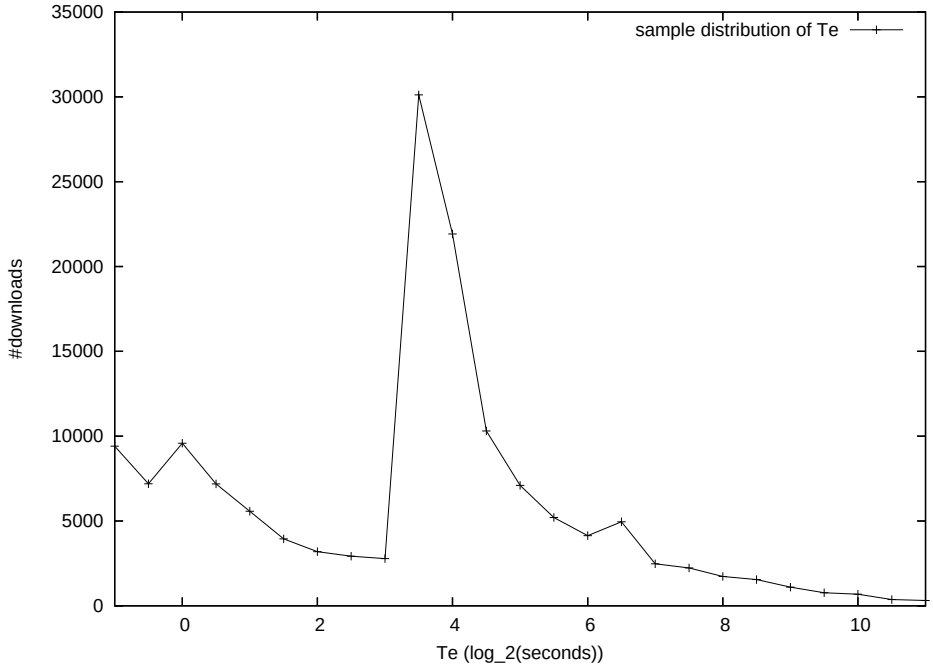


Figure 3.7: Samples distribution of P(Te) relations using download as basic unit.

3.3 Constructing simulator

Comparing different relations that we have obtained above, we find that the $P(T)$ and $P(Te)$ relations have a more monotonic trend than $P(k)$s. We finally pick Te as congestion measurement parameter which is to be used in our simulator because we need to predict which downloads would abort when they start (more explanation on this point later).

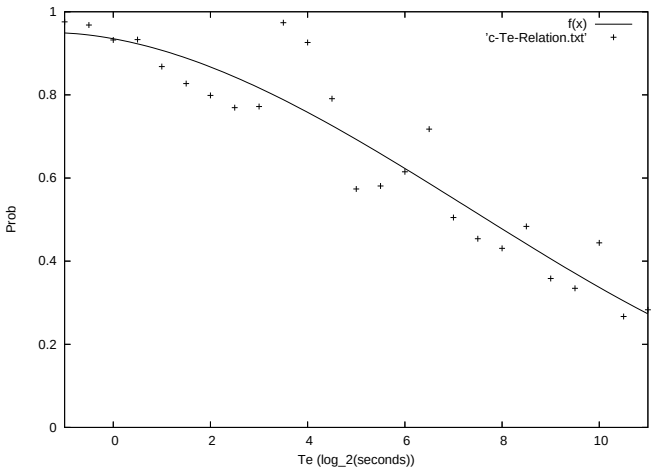
The learning procedure and results are presented below. First we calculate expected download time with the formula $T_e = T_{down} * S_{expect}/S_{down}$. The three parameters (downloaded time, expected download size, and downloaded size) can all be retrieved from the trace summary. S_{expect} (expected download size) is just the sum of file sizes in a web page that can be retrieved directly from the trace, whose way of adjusting has been introduced in section3.1. Then we categorize expected download time into bins by length (using log scale), put the downloads into bins, calculate the mean values of the three probabilities of each bin, and then we will have a curve for every P(T_e) function, namely, Pc(T_e), Pn(T_e), and Pr(T_e). After that we fit them with one polynomial function for each. Those fitted

functions are going to be used in our simulator as default functions for calculating the three probabilities. The learned $P(T)$ curves are summarized in Fig.3.8, in which the fitted function is $f(x) = a * x^3 + b * x^2 + c * x + d$. The rationale behind the choice of fitting function is: 1) We expect two turns in the curves: to drop when Te increases to a certain value, and to converge when Te approximates infinity. 2) The points in the P(Te)s look like a polynomial function. 3) We want the function to be simple as long as it can fit well. Based on these points, we choose the cubic function as the fitting model, which turns out to fit quite well.

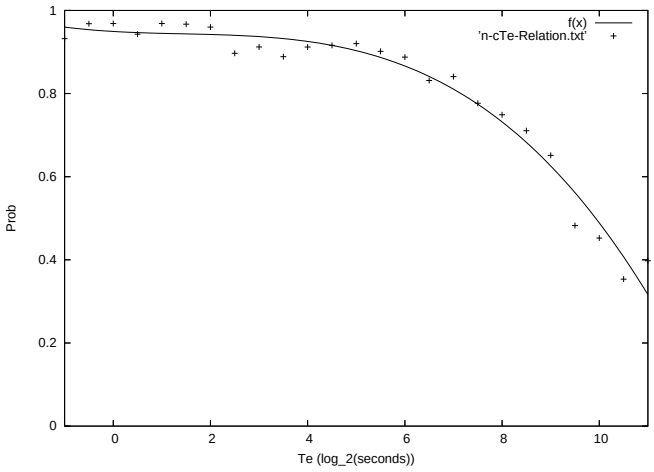
The relations that we have mentioned above are not enough to imitate user behavior, because knowing the probabilities can only decide which downloads to abort, click-next, or retry. But we cannot just kill the downloads from when they started, since every aborted download still has its life time in which it will consume the related network resources in real network. This can be described as a "when to check and when to kill" problem.

Our method is to used Pc(Te) (Pc is the probability of complete, Te is the expected download time) relation to decide which downloads will be killed. The check time is just before the starting of every download. Meanwhile, we calculate the abort time for every download that is sentenced to death with the relation Ta(Te) (Ta is the abort time of the download), which is also learned from our trace summary. The learned curve and its fitted function $g(x) = a * x + b$ are illustrated in Fig.3.9.

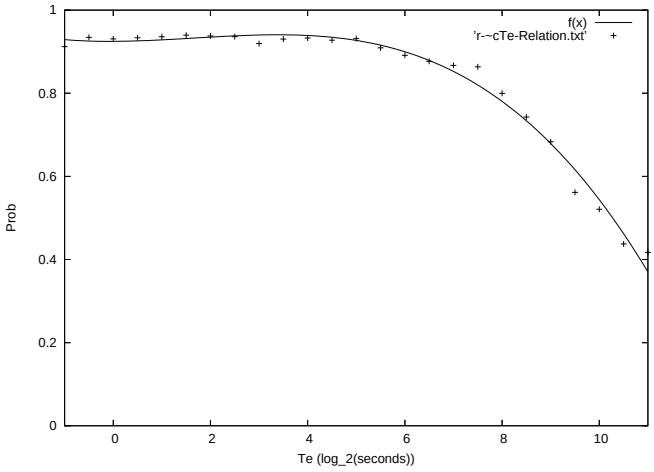
Using this methodology, we develop a surfer simulator based on the most popular network simulator – NS2.



(a) Pc(Te)



(b) Pn(Te)



(c) Pr(Te)

Figure 3.8: P(T_e) relations

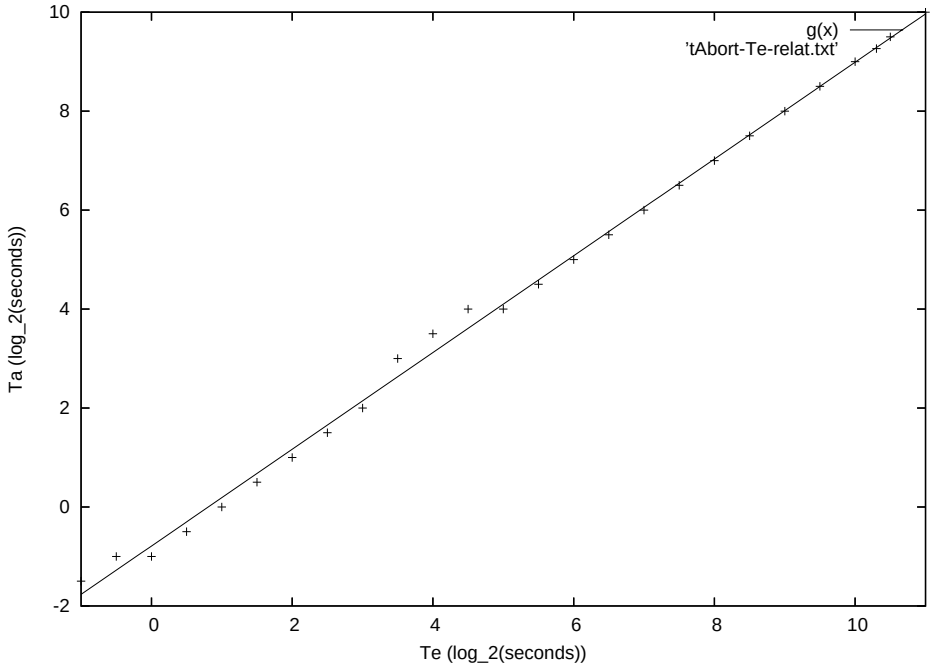


Figure 3.9: The relation between abort time and expected download time, with the fitted function g(x)

Chapter 4

Studying Effect of User Behavior with the Surfer Simulator

Now we are able to study the effects of user behavior in the field of network traffic modeling with our simulator. One big topic that comes in hand is RED (random early detection). This technique has been discussed and deployed for many years. Yet people are still not sure about its best configuration, or even its effect. Consequently, a lot of research has been done for RED's performance on different specific applications.

4.1 Christiansen et al.'s related work

In 2001, Christiansen et al. did a wide evaluation on tuning RED parameters for Web traffic in his then widely referenced paper [14]. However, their emulation research uses Mah's web traffic model [10] which completely ignores congestion-induced user behavior. Therefore, a review of Christiansen et al.'s research is meaningful.

In this chapter, we are going to introduce our comparison experiments and results contradicting those of Christiansen's. Before that, a general view to Christiansen's experiment methodology and results may be deserved.

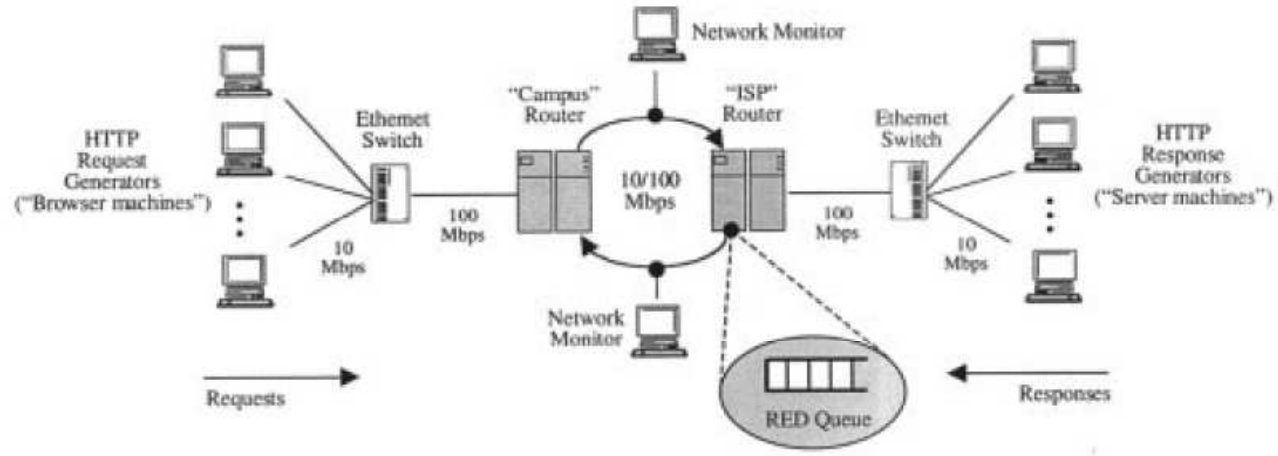


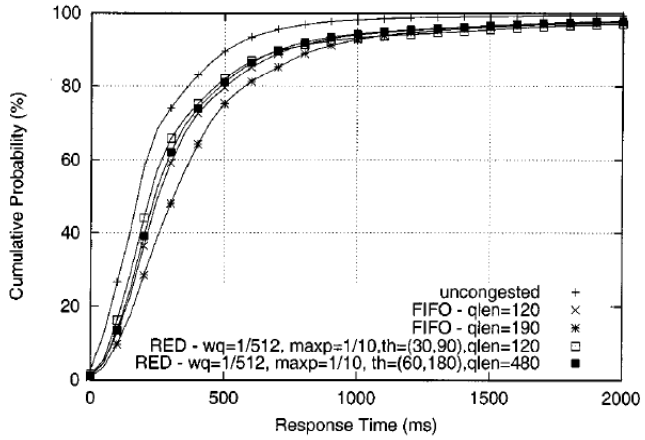
Figure 4.1: Topology of Christiansen's emulation network

4.1.1 Christiansen's experiment methodology and some relevant results

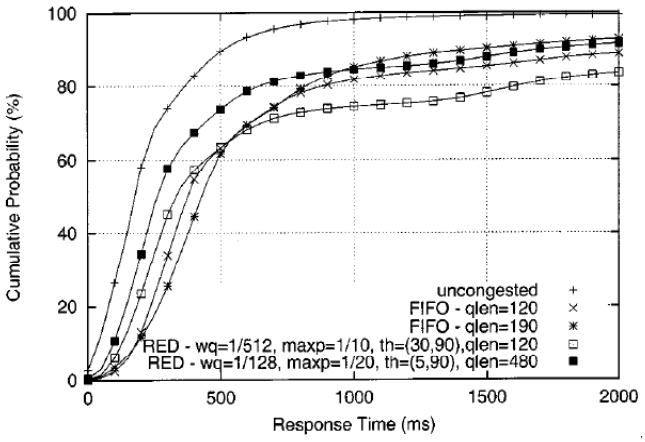
As specified in [14], Christiansen et al. construct an emulation network with a dumbbell topology, a set of machines running web request generator at one side, and another set of machines running web response generator at the other side. Both clients and servers are not real applications but traffic emulators. Two routers sit in either side of dumbbell as gateways. And the network monitors sit in the middle of the bottleneck link between routers. (see Fig.4.1 [14])

The traffic generators are implemented using Mah's web traffic model which has been introduced before. The major response time performance comparison between Drop-Tail and RED is shown in Fig.4.2.

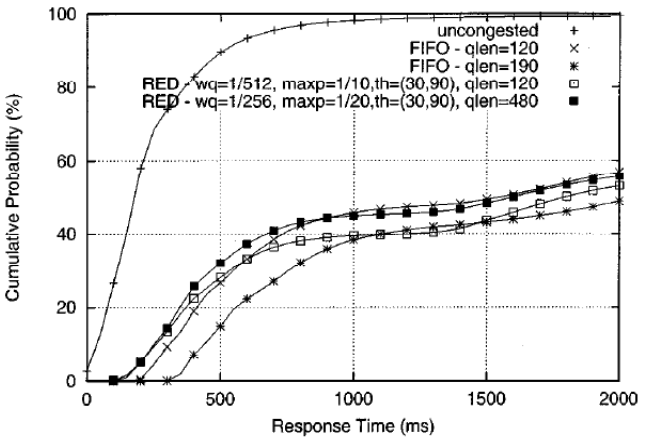
The main conclusions by Christiansen et al. are: (1) Compared to a FIFO queue, RED has a minimal effect on HTTP response times for offered loads up to 90% of link capacity; (2) response times at loads in this range are not substantially affected by RED parameters; (3) between 90% and 100% load, RED can be carefully tuned to yield performance somewhat superior to FIFO; and (4) in such heavily congested networks, RED parameters that provide the best link utilization produce poorer response times. Their final judgement is that RED



(a)



(b)



(c)

Figure 4.2: Response Time Performance Comparison: (a)FIFO and RED at 90% offered load. (b)FIFO and RED at 100% offered load. (c)FIFO and RED at 110% offered load.

queue management is generally useless for links carrying Web traffic.

4.2 Experimental methodology

We decide to check the effects of user behavior by doing similar experiments and compare our results with Christiansen's famous paper [14].

4.2.1 Topology

We use dumbbell topology in the simulation setting as Christiansen did (shown in Fig.4.1), putting the same number of clients and servers in each side, and one switch/router as gateway for each side.

In HTTP level, we configure those end nodes as client/server pairs at the start time. Namely, one client matches one exact server. Since our simulation focuses on a congestion period of several minutes, this simplification should be acceptable. Besides, what we care about here is the overall performance of queuing methods in gateway routers for a bottleneck link. Which server a client chooses should not affect our results. Also, in our simulation, one click only initiates one download, and corresponding server will open a new connection for every new download (HTTP 1.0), as the setting of Christiansen's.

Since we use open model (the surfer session model that we introduced in chapter 2 is an open model) in the researches here, we can use session arrival rate to tune the loads of the network.

4.2.2 Parameter settings

Some important parameters are listed here. Bottleneck bandwidth is set to 1Mb, so that we can achieve high load with relatively less clients. Download size uses Pareto distribution, with shape parameter 1.2 and a mean value of 110KB retrieved from our trace. Think time uses Weibull distribution, with scale=60, and shape=0.5. This is from [15] which focuses on

the research of modeling HTTP request arrivals. Thus the mean think time is about 120s.

The page model we use is Pagepool/Math (see NS2 manual for details) for simplicity, that is, one download has only one object. The simulation results may be slightly different from those using compound page model. But the major implications of the experiments should still hold. The bandwidth from end nodes to router is set to 100Mb to guarantee that the only bottleneck is the link between two gateway routers.

The following parameters are configured the same with common settings from Christiansen's paper [14]. RED parameters group 1: $q_{length} = 120$, $min_{thresh} = 30$, $max_{thresh} = 90$, $weight_q = 1/512$, $max_{prob} = 1/10$. RED parameters group 2: $q_{length} = 480$, $min_{thresh} = 5$, $max_{thresh} = 90$, $weight_q = 1/128$, $max_{prob} = 1/20$.

The simulation of every case below is 6000 seconds, ignoring the first half to guarantee that the traffic has entered steady state, i.e. we start measuring from the second half of the simulation.

4.3 Experiment procedure

First, we need to have a load calibration so as to control the loads. More specifically, what we control is the offered loads, i.e. the maximum loads that the clients who are under our setting can generate when there is no frustration. When congestion-induced user behavior exists, the real loads in the network would be much smaller/lighter than the offered loads. We obtain offered loads in the way of Christiansen's. Set enough bottleneck bandwidth (increase from 1Mb to 10Mb) and run some simulations with different session arrival rates. Thus we get a relation between offered loads and the session arrival rate in Fig.4.3. From this curve we can see that arrival rate of 0.08, 0.09 and 0.1 approximately produce 80%, 90% and 100% offered loads respectively. With user behavior on, we have about 105 sessions running when traffic is stable. Next we can try different loads according to the calibration on different settings of parameters.

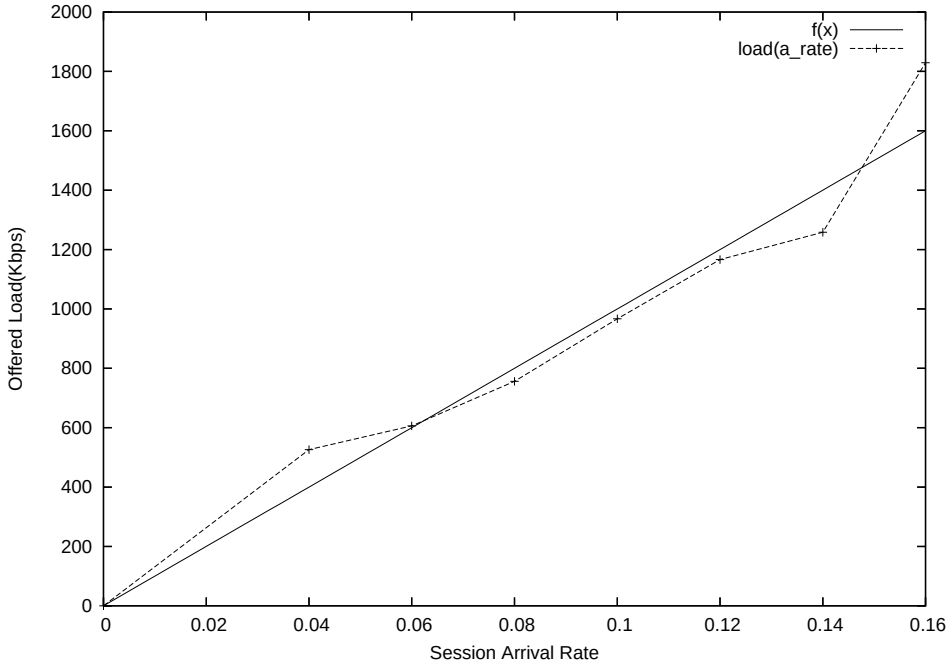


Figure 4.3: Session Arrival Rate vs. Offered Load

We tried the cases of arrival rate ranging from 0.05 to 0.15 (offered loads of about 50% to 150% of the capacity), and found a lot of differences with those from [14] showing the important effects of user back-off, as discussed in next section.

4.4 Experiment results and implications

This section presents and discusses our experiment results.

4.4.1 Some explanation on the result figures

The resulting figures from the simulation are shown in Fig. 4.6, 4.7 and 4.8. In the figures, "DropTail" stands for FIFO case; "RED" stands for RED case; and "RT" means response time. The vertical straight lines in the figures that are tabbed as "avg *something*" is the average value of "*something*". Note that in some figures you may see a straight zero line or may not see the average value plotted. That is because the values of response times in those cases are too long to be displayed in 10 seconds range for the response time.

4.4.2 The results and implications

Now we can study Fig. 4.6 and 4.8 to see if the conclusions in [14] can still hold with user behavior counted in.

One important conclusion of [14] is that when offered loads are not more than 90% of the bottleneck bandwidth, RED provides minimal effects on HTTP response times, i.e. it is no better than FIFO. On the contrary, in Fig.4.6(c) and Fig.4.8(c), both of which provide about 80% offered loads, the differently set RED both provide much shorter response times.

Another conclusion of [14] is that when loads are near 100% of the link capacity, tuned RED should beat FIFO in response time performance. But on the contrary, Fig.4.6(d), 4.6(e), 4.8(d), and 4.8(e), (which provide loads from 90% to 100%) show that FIFO works better in this range. Note that the two groups of RED settings that we use here are those who are supposed to beat FIFO in this range of offered loads in [14].

Another interesting point that we can see from Fig.4.6(f) and Fig.4.8(f) is that even when offered loads increase to 150% of the capacity, RED can be tuned to give much better response time performance.

For a cleaner view of how performance varies with user backoff, we plot arrival rate versus mean response time of Fig.4.6 and Fig.4.8 in Fig.4.4 and Fig.4.5. And the following are my implications from them. Since our research focus is the performance difference caused by user backoff, namely, the congested area, we do not provide a complete proof for it here.

When offered load is light, there are seldom traffic peaks; early packet drop is not necessary; thus drop-tail has similar performance with RED. As traffic grows from 50% to 80%, the congestion is still light, user abort rate is still not high. Thus we see drop-tail's response time increases gradually, while RED's performance is stabilized by early packet drop. The performance of RED in 70% load is even a little better than that of 50%, which might be caused by the backoffs of large downloads due to the frustration triggered by some early packet drop. When load increases to 90% of the link capability, users face heavy congestion in the drop-tail case and abort rate increase dramatically; as the large downloads are

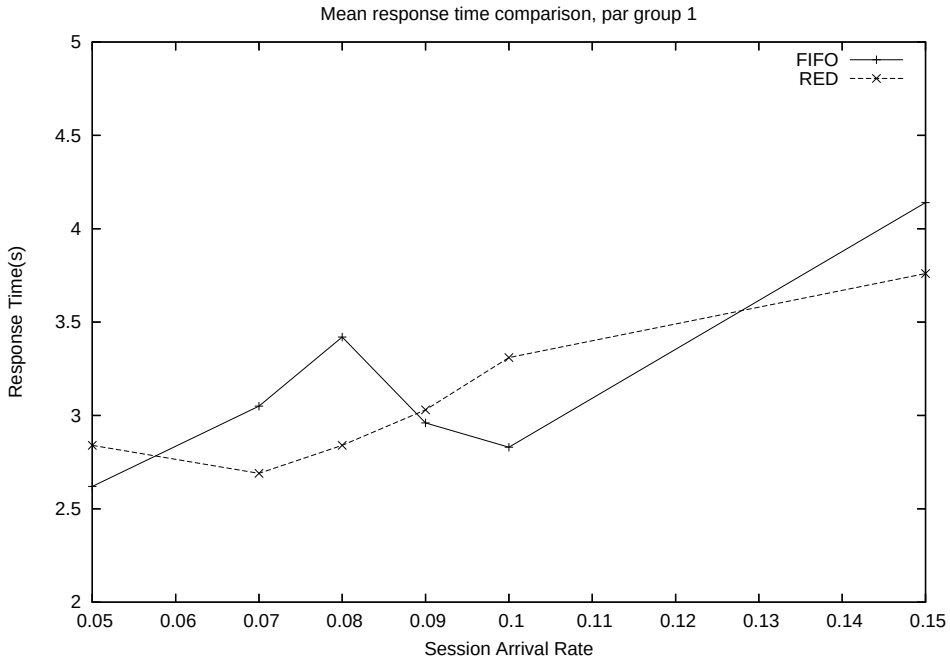


Figure 4.4: Session Arrival Rate vs. Mean Response Time, using three P(Te) functions, RED parameters: $q_{length} = 120$, $min_{thresh} = 30$, $max_{thresh} = 90$, $weight_q = 1/512$, $max_{prob} = 1/10$

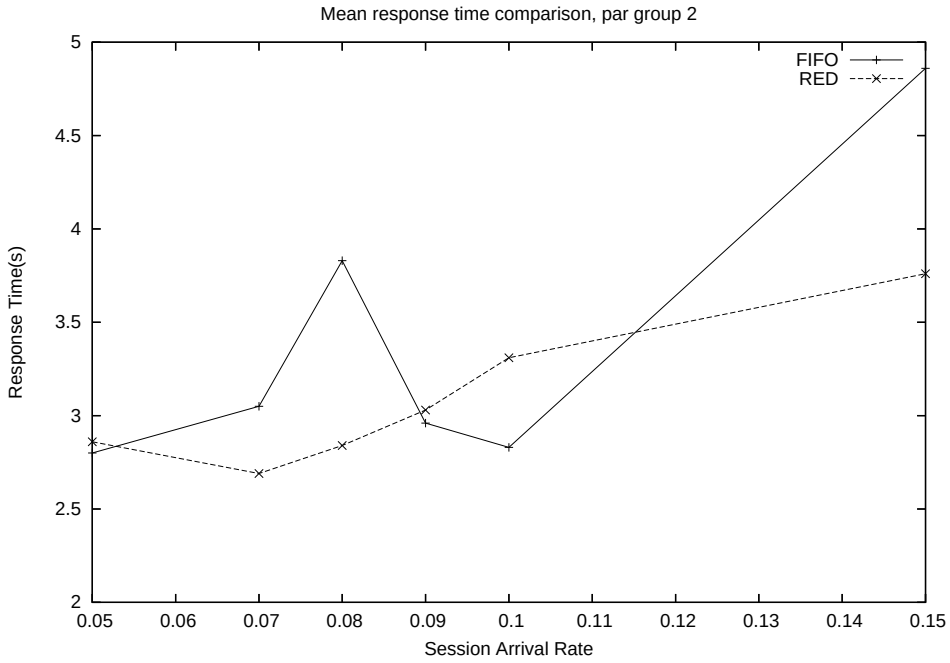


Figure 4.5: Session Arrival Rate vs. Mean Response Time, using three P(Te) functions, RED parameters group2: $q_{length} = 480$, $min_{thresh} = 5$, $max_{thresh} = 90$, $weight_q = 1/128$, $max_{prob} = 1/20$

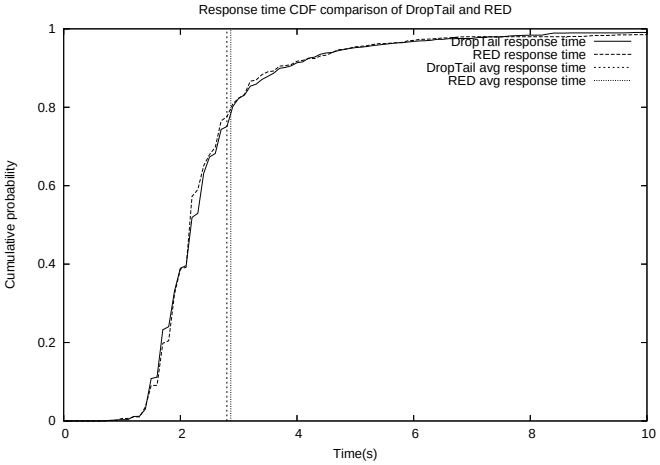
aborted, mean response time in drop-tail case decreases in great deal. Since user's abort rate will not increase infinitely, drop-tail's response time increases again when the offered load goes too high.

In addition, to verify whether the above differences are caused by the congestion-induced user behavior. We conduct another batch of experiments with user reaction turned off. We using RED parameter group 1, arrival rate ranging from 0.05 to 0.15, but with a sequence of fixed probabilities (instead of the P(Te) functions that we have learned: $P_{abort} = 0.05$, $P_{next} = 0.94$, $P_{retry} = 0.92$, which are near those of the average case of our trace. This configuration is close to the model that is used in [14].

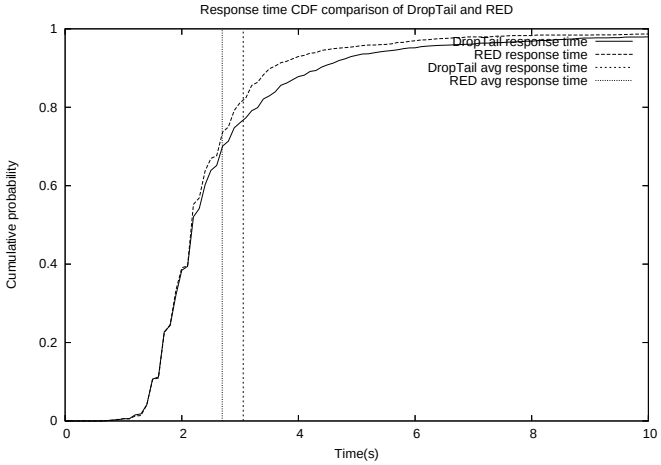
The results, which are presented in Fig.4.7, show that the conclusions of [14] become true in this situation. For example, Fig.4.7(a) and Fig.4.7(c) show that RED has no better and even worse response time performance than FIFO when offered loads are less than 90% of link capacity; Fig.4.7(e) shows that RED is tuned to perform better than FIFO in the load range near 98% of the capacity.

4.5 Conclusion

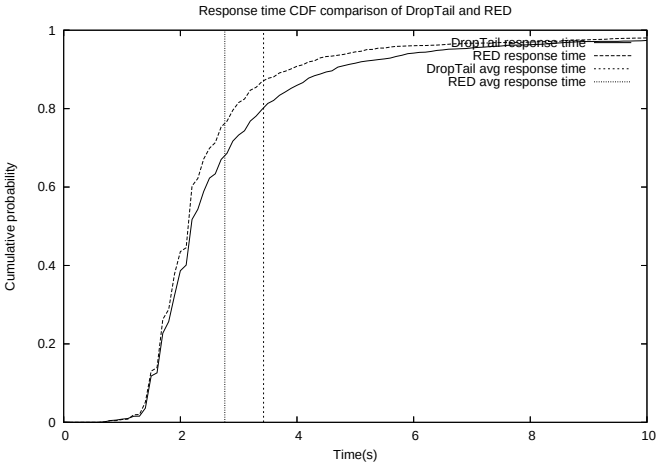
In this thesis, we examined the surfer model proposed by Tay et al., and then introduced our procedure to learn surfer-behavior related mathematical relations from network traces, after that we discussed how we develop surfer simulator to imitate the user behavior. Finally, we tried our simulator to study the RED tuning for the web traffic, and justified (with the contratry results) that ignoring congestion-induced user behavior could mislead the researchers to incorrect conclusions.



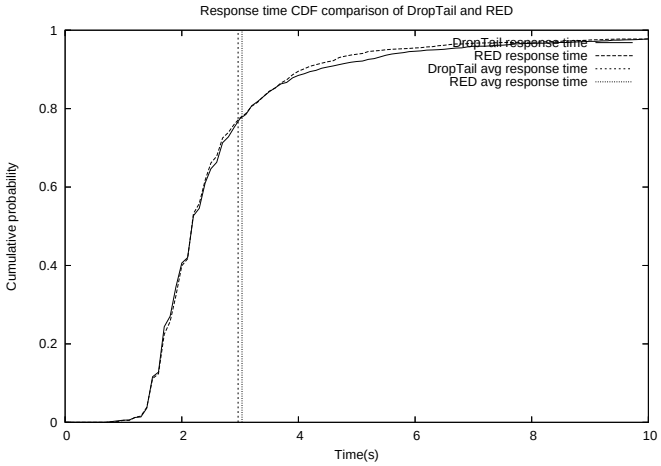
(a) session arrival rate=0.05



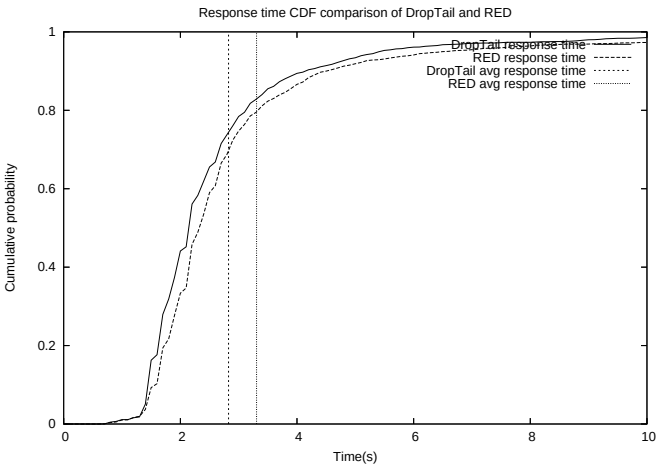
(b) session arrival rate=0.07



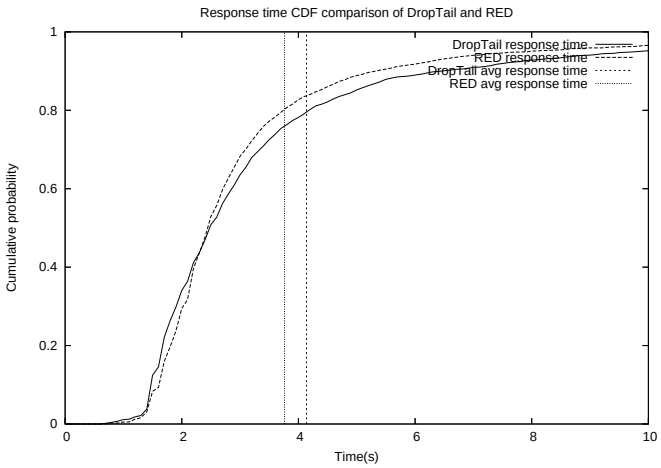
(c) session arrival rate=0.08



(d) session arrival rate=0.09

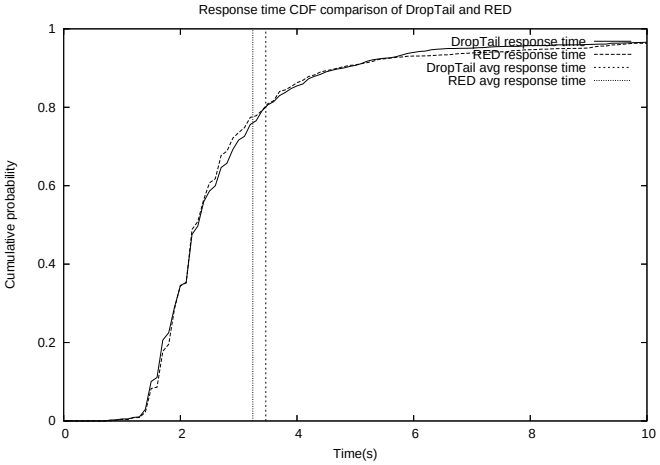


(e) session arrival rate=0.10

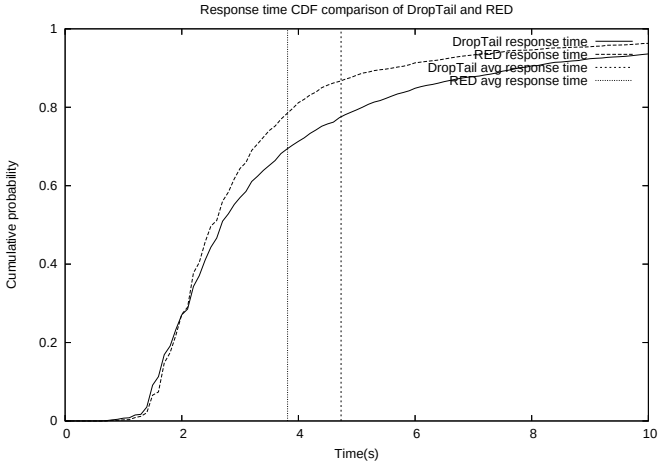


(f) session arrival rate=0.15

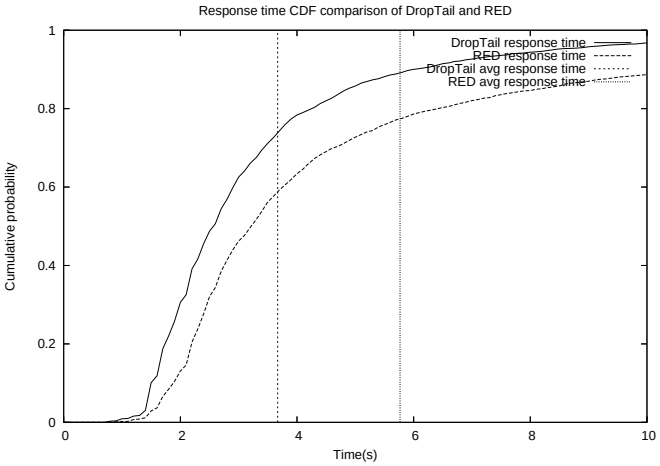
Figure 4.6: Response time performance comparison of different session arrival rates, using three P(Te) functions, RED parameters: $q_{length} = 120$, $min_{thresh} = 30$, $max_{thresh} = 90$, $weight_q = 1/512$, $max_{prob} = 1/10$



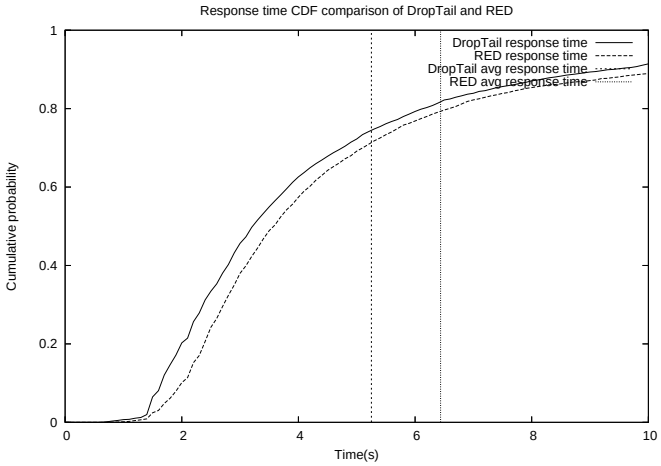
(a) session arrival rate=0.05



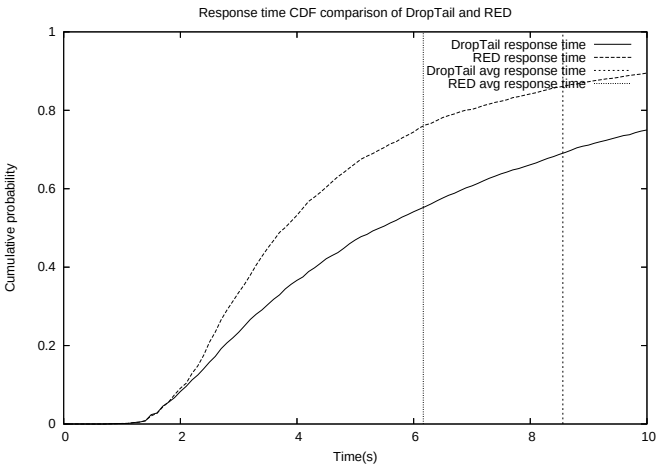
(b) session arrival rate=0.07



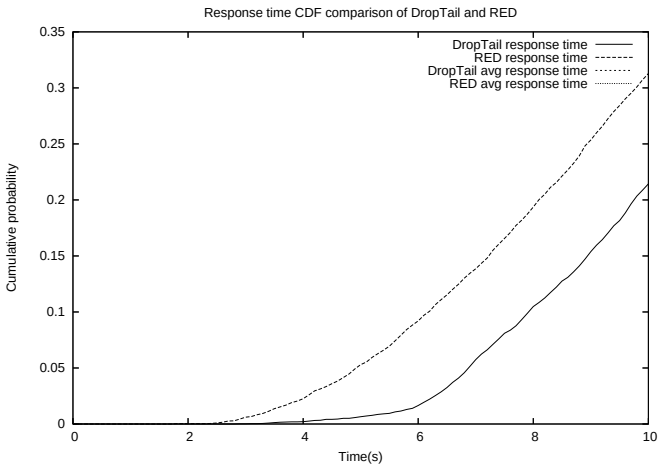
(c) session arrival rate=0.08



(d) session arrival rate=0.09

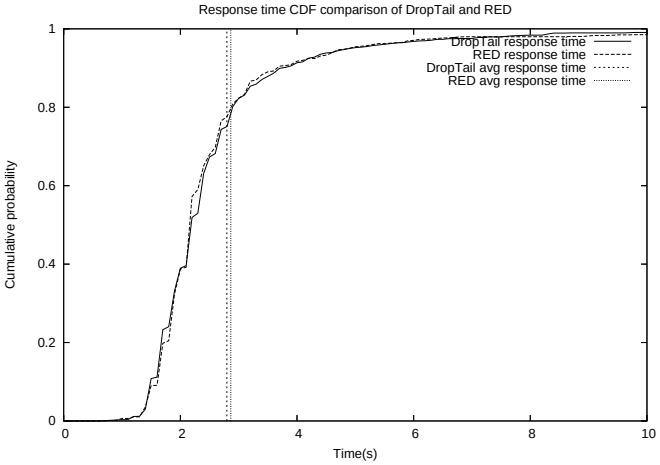


(e) session arrival rate=0.10

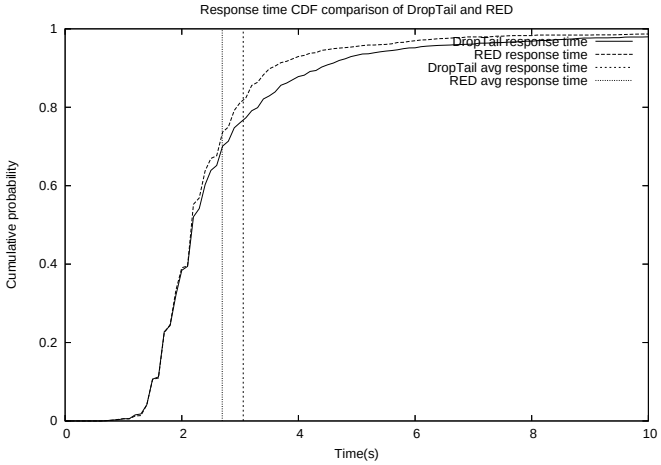


(f) session arrival rate=0.15

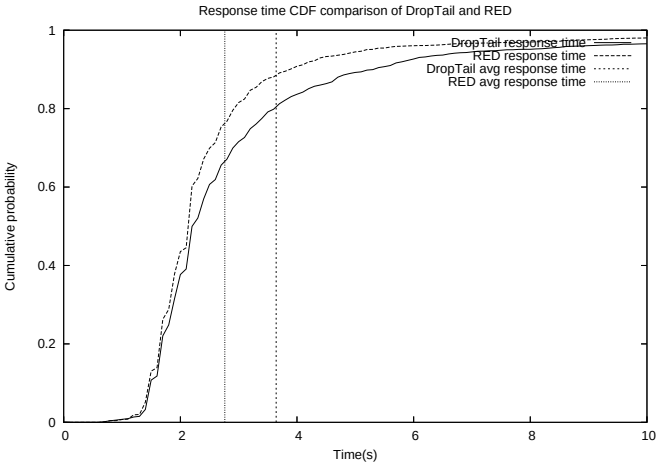
Figure 4.7: Response time performance comparison of different session arrival rates, using fixed Pa, Pn and Pr, RED parameters: $q_{length} = 120$, $min_{thresh} = 30$, $max_{thresh} = 90$, $weight_q = 1/512$, $max_{prob} = 1/10$



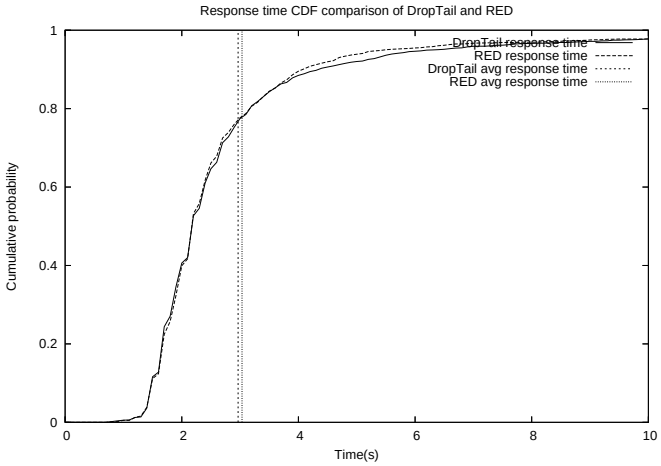
(a) session arrival rate=0.05



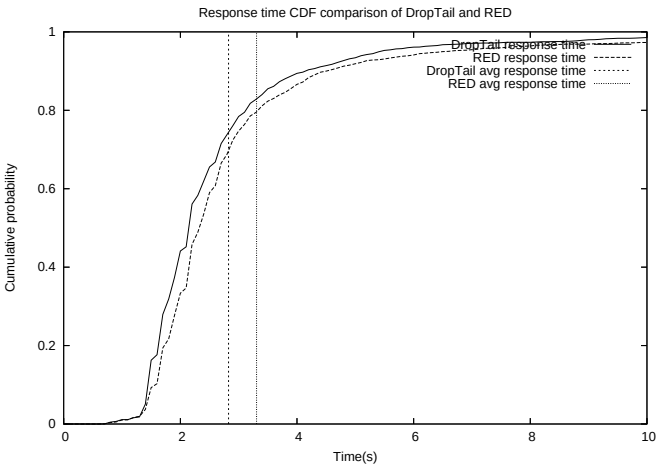
(b) session arrival rate=0.07



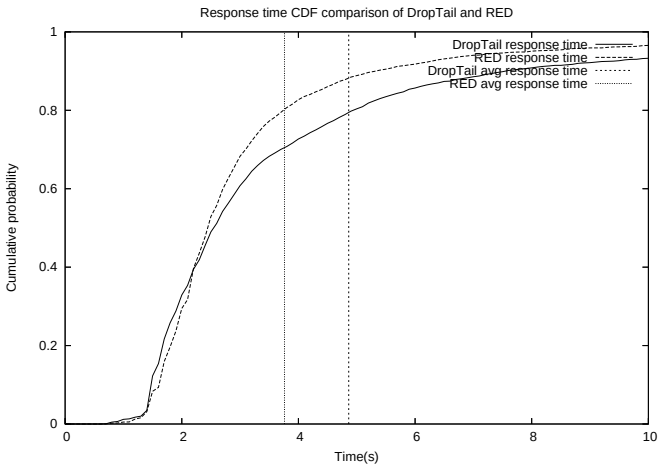
(c) session arrival rate=0.08



(d) session arrival rate=0.09



(e) session arrival rate=0.10



(f) session arrival rate=0.15

Figure 4.8: Response time performance comparison of different session arrival rates, using three P(Te) functions, RED parameters group2: $q_{length} = 480$, $min_{thresh} = 5$, $max_{thresh} = 90$, $weight_q = 1/128$, $max_{prob} = 1/20$

Chapter 5

Future work

Imitating user behavior is difficult. We look forward to learning more accurate $P(T_e)$ and $T_{abort}(T_e)$ from traces in future. The key might be finding a way to obtain accurate expected download size.

Since our comparison with Christiansen's results has revealed considerable difference between with and without congestion-induced user behavior cases, we believe there should potentially be a batch of researches that may have been misled by ignoring user back-off. We look forward to more impacts by introducing user behavior to existing literature.

Besides, currently the surfer and demand/supply models (see [12]) are still generally qualitative models, many distributions and relations between parameters need to be found to make them a quantitative models (which may not be easy). Once this is achieved, a more accurate traffic generator can also be produced.

Bibliography

[1] B. Mandelbrot. Self-similar error clusters in communication systems and the concept of conditional stationarity. *IEEE Trans. Communication Technology*, pages 71–90, 1965.

[2] H. Fowler and W. Leland. Local area network traffic characteristics with implications for broadband network congestion mangement. *IEEE J. Select. Ar. Comm.*, 9(7):1139–1149, 1991.

[3] C. Meier-Hellstern, P. Wirth, Y. Yan, and D. A. Hoeflin. Traffic models for isdn data users: office automation application. *ITC-13*, pages 167–172, 1991.

[4] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, February 1994.

[5] R. Sherman, M. S. Taqqu, and W. Willinger. Proof of a fundamental result in self-similar traffic modeling. *Computer Communications Review*, 1998.

[6] Carl Nuzman and et al. A compound model for tcp connection arrivals. *Computer Networks*, March 2000.

[7] Vern Paxson and Sally Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Trans. on Networking*, 3(3):226–244, June 1995.

[8] Anja Feldmann. Characteristics of tcp connection arrivals. *unpublished*, December 1998.

[9] Henrik Abrahamsson and Bengt Ahlgren. Using empirical distributions to characterize web client traffic and to generate synthetic traffic. In *GLOBECOM (1)*, pages 428–433, Nov. 2000.

[10] Bruce A. Mah. An empirical model of HTTP network traffic. In *INFOCOM (2)*, pages 592–600, 1997.

[11] Hyoung-Kee Choi and John O. Limb. A behavioral model of web traffic. In *Int. Conf. Network Protocols*, pages 327–334, Oct. 1999.

[12] Y. C. Tay, Dinh Nguyen Tran, Eric Yi Liu, Wei Tsang Ooi, and Robert Morris. Modeling web surfers and bandwidth demand/supply for congestion-induced behavior. *Submitted*, May 2006.

[13] D.N. Tran, W.T. Ooi, and Y. C. Tay. Sax: A tool for studying congestion-induced surfer behavior. `http://www.pam2006.org/program.html/`, Mar. 2006.

[14] M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith. Tuning red for the web traffic. *IEEE/ACM Transactions on Networking*, 9(3), June 2001.

[15] K. H. Yeung and C. W. Szeto. On the modeling of www request arrivals. *IEEE ICPPW*, 1999.